Finite state machine

From Wikipedia, the free encyclopedia
(Redirected from State machine)

A **finite state machine** (FSM) or **finite automaton** is a model of behavior composed of states, transitions and actions. A state stores information about the past, i.e. it reflects the input changes from the system start to the present moment. A transition indicates a state change and is described by a condition that would need to be fulfilled to enable the transition. An action is a description of an activity that is to be performed at a given moment. There are several action types:

Entry action
: execute the action when entering the state

Exit action
: execute the action when exiting the state

Input action
: execute the action dependent on present state and input conditions

Transition action
: execute the action when performing a certain transition

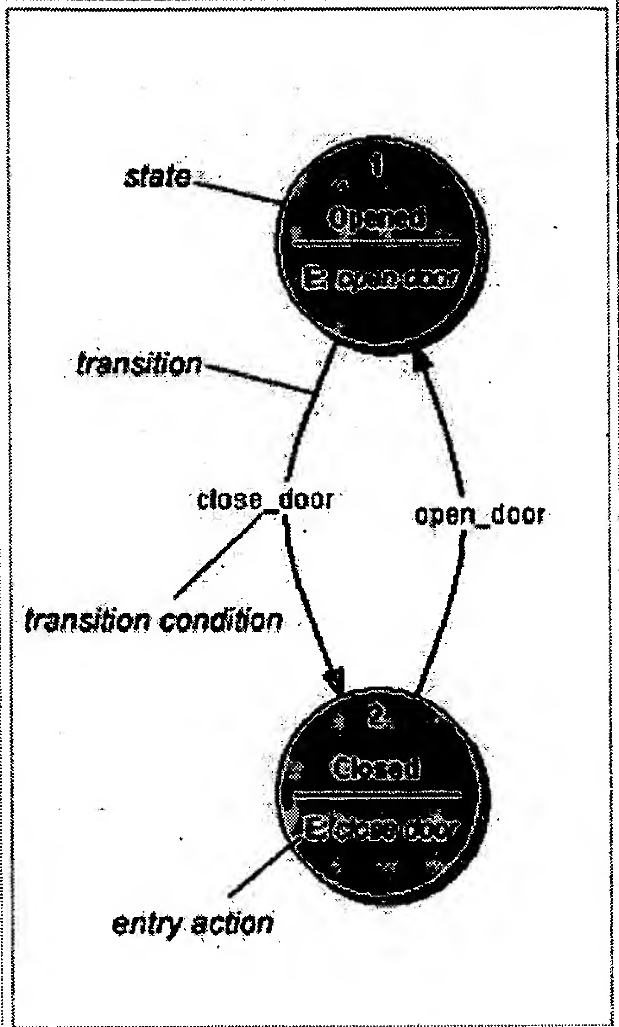

Fig.1 Finite State Machine

FSM can be represented using a state diagram (or state transition diagram) as in figure 1. Besides this, several state transition table types are used. The most common representation is shown below: the combination of current state (B) and condition (Y) shows the next state (C). The complete actions information can be added only using footnotes. An FSM definition including the full actions information is possible using state tables (see also VFSM).

State transition table

Current State/ Condition	State A	State B	State C
Condition X	...	...	...
Condition Y	...	State C	...
Condition Z	...	...	...

In addition to their use in modeling reactive systems presented here, finite state automata are significant in many different areas, including linguistics, computer science, philosophy, biology, mathematics, and logic. A complete survey of their applications is outside the scope of this article. Finite state machines are one type of the automata studied in automata theory and the theory of computation. In computer science, finite state machines are widely used in modelling of application behaviour, design of hardware digital systems, software engineering, compilers, and the study of computation and languages.

Contents

Classification

There are two groups distinguished: Acceptors/Recognizers and Transducers.

Acceptors and recognizers

This kind of machine gives a binary output, saying either *yes* or *no* to answer whether the input is accepted by the machine or not. The machine can also be described as defining a language, in this case the language defined would contain every word accepted by the machine but none of the rejected ones. All states of the FSM are said to be either accepting or not accepting. If when all input is processed the current state is an accepting state, the input is accepted, otherwise not. As a rule the input are symbols (characters); actions are not used. The example in figure 2 shows a finite state machine which accepts the word "nice", in this FSM the only accepting state is number 7.

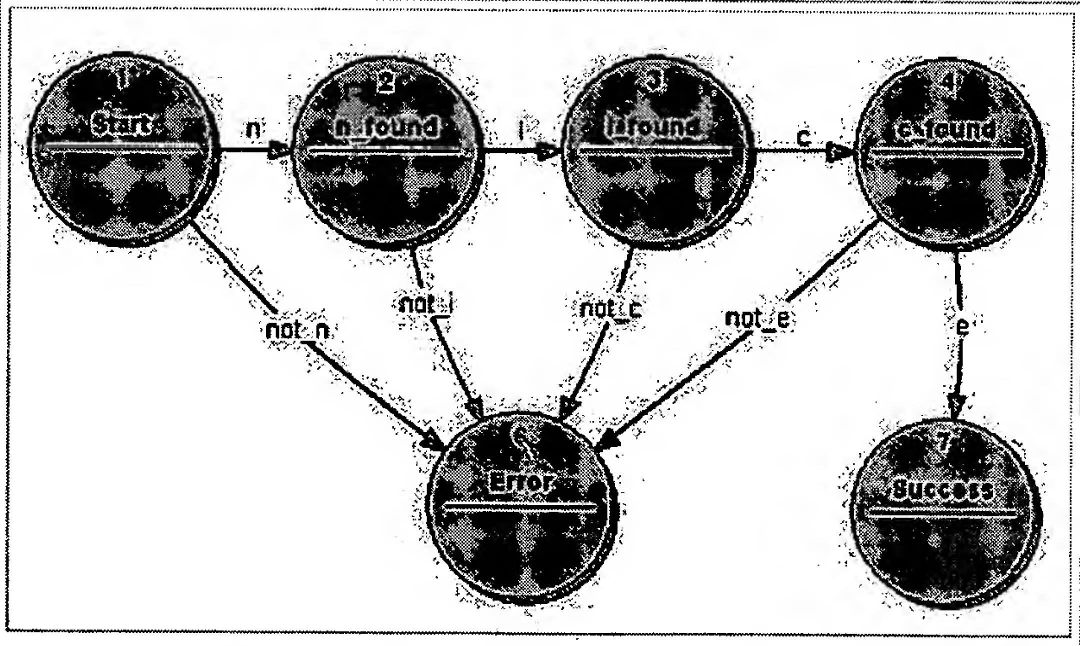

Fig. 2 Acceptor FSM: parsing the word "nice"

Transducers

Transducers generate output based on a given input and/or a state using actions. They are used for control applications. Here two types are distinguished:

Moore machine
: The FSM uses only entry actions, i.e. output depends only on the state. The advantage of the Moore model is a simplification of the behaviour. The example in figure 3 shows a Moore FSM of an elevator door. The state machine recognizes two commands: "command_open" and "command_close" which trigger state changes. The entry action (E:) in state "Opening" starts a motor opening the door, the entry action in state "Closing" starts a motor in the other direction closing the door. States "Opened" and "Closed" don't perform any actions. They signal to

the outside world (e.g. to other state machines) the situation: "door is open" or "door is closed".

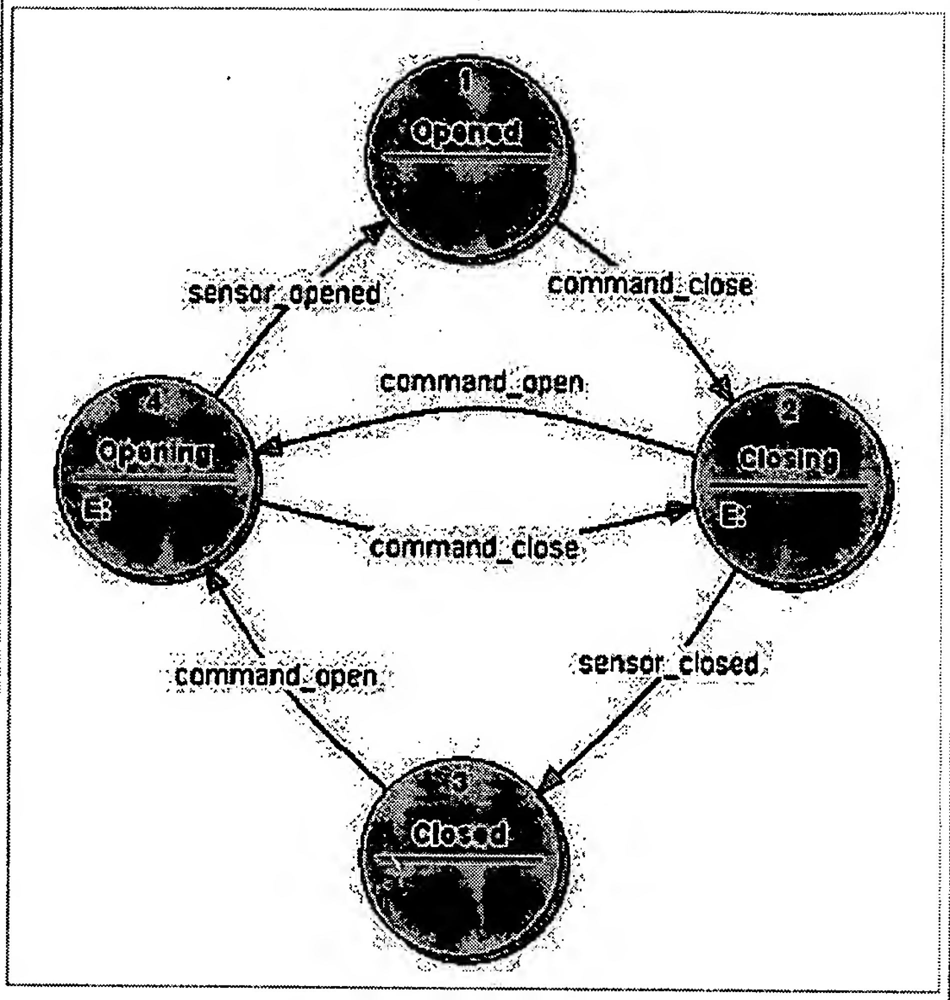

Fig. 3 Transducer FSM: Moore model example

Mealy machine

The FSM uses only input actions, i.e. output depends on input and state. The use of a Mealy FSM leads often to a reduction of the number of states. The example in figure 4 shows a Mealy FSM implementing the same behaviour as in the Moore example (the behaviour depends on the implemented FSM execution model and will work e.g. for virtual FSM but not for event driven FSM). There are two input actions (I:): "start motor to close the door if command_close arrives" and "start motor in the other direction to open the door if command_open arrives".

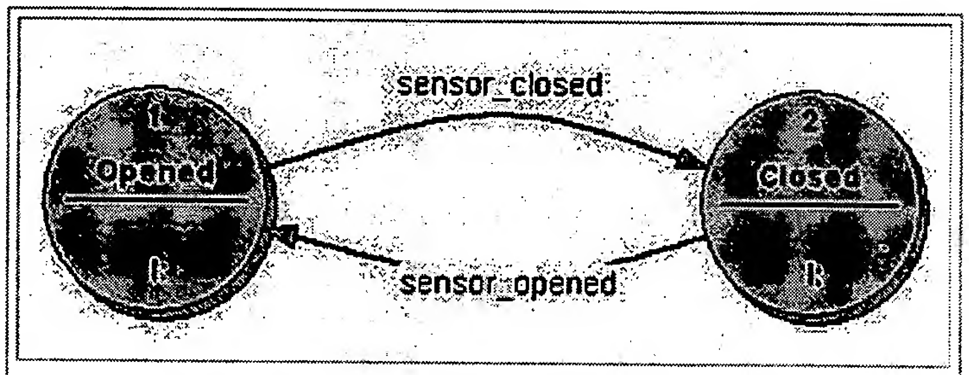

Fig. 4 Transducer FSM: Mealy model example

In practice mixed models are often used.

More details about the differences and usage of Moore and Mealy models, including an executable example, can be found in the external technical note "Moore or Mealy model?" (http://www.stateworks.com/active/content/en/technology/technical_notes.php#tn10)

A further distinction is between **deterministic** (DFA) and **non-deterministic** (NDFA, GNFA) automata. In deterministic automata, for each state there is exactly one transition for each possible input. In non-deterministic automata, there can be none or more than one transition from a given state for a given possible input. This distinction is relevant in practice, but not in theory, as there exists an algorithm which can transform any NDFA into an equivalent DFA, although this transformation typically significantly increases the complexity of the automaton.

The FSM with only one state is called a combinatorial FSM and uses only input actions. This concept is useful in cases where a number of FSM are required to work together, and where it is convenient to consider a purely combinatorial part as a form of FSM to suit the design tools.

FSM logic

The next state and output of a FSM is a function of the input and of the current state. The FSM logic is shown in Figure 5

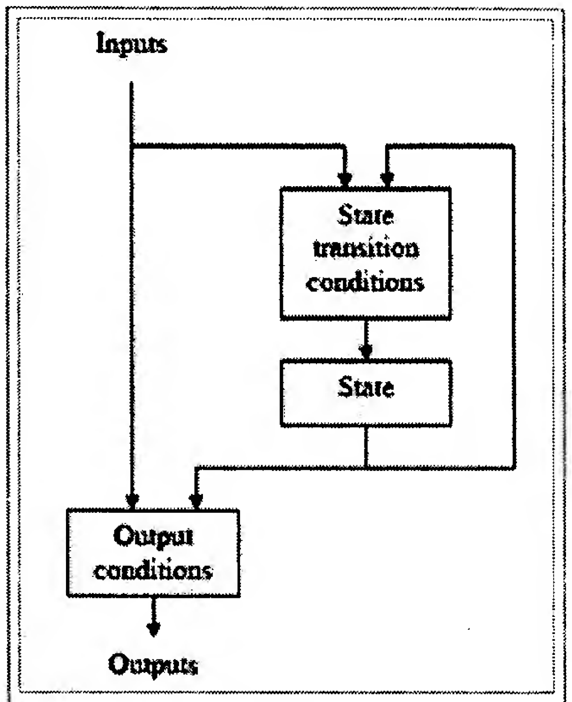

Fig. 5 FSM Logic

Mathematical model

Depending on the type there are several definitions. An **acceptor** finite state machine is a quintuple <Σ, S, s_0, δ, F>, where:

- Σ is the input alphabet (a finite non empty set of symbols).
- S is a finite non empty set of states.
- s_0 is an initial state, an element of S. In a Nondeterministic finite state machine, s_0 is a set of initial states.
- δ is the state transition function: δ: $S \times \Sigma \rightarrow S$.
- F is the set of final states, a (possibly empty) subset of S.

A **transducer** finite state machine is a sextuple <Σ, Γ, S, s_0, δ, ω>, where:

- Σ is the input alphabet (a finite non empty set of symbols).
- Γ is the output alphabet (a finite non empty set of symbols).
- S is a finite non empty set of states.
- s_0 is the initial state, an element of S. In a Nondeterministic finite state machine, s_0 is a set of initial states.
- δ is the state transition function: δ: $S \times \Sigma \rightarrow S \times \Gamma$.
- ω is the output function.

If the output function is a function of a state and input alphabet (ω: $S \times \Sigma \rightarrow \Gamma$) that definition corresponds to the **Mealy model**. If the output function depends only on a state (ω: $S \rightarrow \Gamma$) that definition corresponds to the **Moore model**.

Optimization

Optimizing an FSM means finding the machine with the minimum number of states that performs the same function. This problem can be solved using a coloring algorithm.

Implementation

Hardware applications

In a digital circuit, a FSM may be built using a programmable logic device, a programmable logic controller, logic gates and flip flops or relays. More specifically, a hardware implementation requires a register to store state variables, a block of combinational logic which determines the state transition, and a second block of combinational logic that determines the output of a FSM.

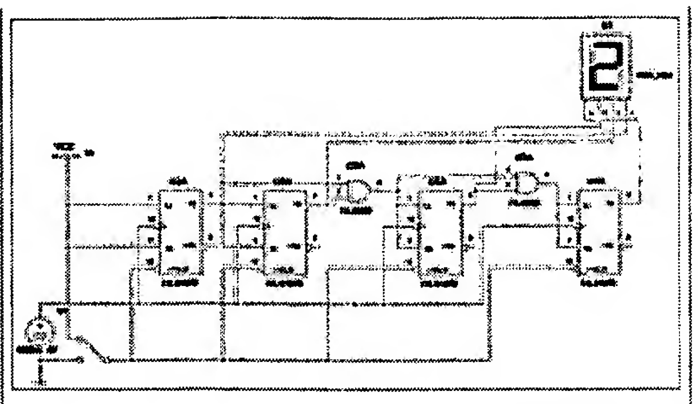
Fig. 6 The circuit diagram for a 4 bit TTL counter, a type of state machine

Software applications

Following concepts are commonly used to build software applications with finite state machines:

- event driven FSM
- virtual FSM (VFSM)

Tools

- AT&T FSM Library™ [1] (http://www.research.att.com/projects/mohri/fsm/)
- AutoFSM [2] (http://autogen.sourceforge.net/autofsm.html)
- AsmL [3] (http://research.microsoft.com/fse/asml/)
- Bandera [4] (http://bandera.projects.cis.ksu.edu/)
- Boost Statechart Library [5] (http://boost-sandbox.sourceforge.net/libs/statechart/doc/index.html)
- CAZE - FSM-based .NET authorization library [6] (http://www.lamarvin.com/caze_default.asp)
- Covered [7] (http://covered.sourceforge.net/)
- Concurrent Hierarchical State Machine [8] (http://homepage.mac.com/pauljlucas/software/chsm/)
- DescoGUI [9] (http://www.s2.chalmers.se/software/desco/)
- Dynamic Attachment Finite State Machine (DAFSM) [10] (http://dmabco.sourceforge.net/)
- Exorciser [11] (http://www.educeth.ch/informatik/exorciser/)
- Finite State Kernel Creator [12] (http://fskc.sourceforge.net/)
- Finite State Machine Editor [13] (http://fsme.sourceforge.net/)
- Finite State Machine Explorer [14] (http://www.belgarath.org/java/fsme.html)
- FIRE Engine, Station and Works [15] (http://www.fastar.org/)
- FSMGenerator [16] (http://fsmgenerator.sourceforge.net/)
- Grail+ [17] (http://www.csd.uwo.ca/Research/grail/)
- FSA Utilities [18] (http://odur.let.rug.nl/~vannoord/Fsa/)
- Libero [19] (http://www.imatix.com/html/libero/index.h
- Java Finite Automata [20] (http://www.ebi.ac.uk/~kirsch/monq-doc/)
- JFLAP [21] (http://www.jflap.org/)
- jrexx-Lab [22] (http://www.karneim.com/jre
- JSpasm [23] (http://jspasm.sourceforge.net/
- Kara [24] (http://www.swisseduc.ch/informatik/karato
- Nunni FSM Generator [25] (http://nunnifsmgen.nunnisoft.ch/en/)
- Petrify [26] (http://www.lsi.upc.edu/~jordicf/petrify/)
- PyFSA [27] (http://osteele.com/software/python/fsa/)
- Qfsm [28] (http://qfsm.sourceforge.net/)
- Quantum-Leaps [29] (http://www.quantum-leaps.com/)

References

- Timothy Kam, *Synthesis of Finite State Machines: Functional Optimization.* Kluwer Academic Publishers, Boston 1997, ISBN 0-7923-9842-4
- Tiziano Villa, *Synthesis of Finite State Machines: Logic Optimization.* Kluwer Academic Publishers, Boston 1997, ISBN 0-7923-9892-0
- Carroll, J., Long, D. , *Theory of Finite Automata with an Introduction to Formal Languages.* Prentice Hall. Englewood Cliffs, 1989.
- Hopcroft, J.E., Ullman, J.D., *Introduction to Automata Theory, Languages and Computation.* Addison - Wesley, 1979.
- Kohavi, Z., *Switching and Finite Automata Theory.* McGraw-Hill, 1978.

- Gill, A., *Introduction to the Theory of Finite-state Machines*. McGraw-Hill, 1962.
- Cassandras, C., Lafortune, S., "Introduction to Discrete Event Systems". Kluwer, 1999, ISBN 0-7923-8609-4.
- Watson, B.W., *Taxonomies and Toolkits of Regular Language Algorithms*. Ph.D dissertation, Eindhoven University of Technology, Netherlands, 1995, ISBN 90-386-0396-7.

See also

- Abstract state machine
- Automata analyzer
- Coverage analysis
- Marvin Minsky
- Petri net
- Protocol development
- Pushdown automaton
- Regular expression
- Regular grammar
- Simulation
- Sequential logic
- Sparse matrix
- Supervisory control theory
- Transition system

External links

- Description from the Free On-Line Dictionary of Computing (http://foldoc.doc.ic.ac.uk/foldoc/foldoc.cgi?query=finite+state+machine)
- NIST Dictionary of Algorithms and Data Structures entry (http://www.nist.gov/dads/HTML/finiteStateMachine.html)
- Hierarchical State Machines (http://www.eventhelix.com/RealtimeMantra/HierarchicalStateMachine.htm)

Automata theory: formal languages and formal grammars

Chomsky hierarchy	**Grammars**	**Languages**	**Minimal automaton**
Type-0	Unrestricted	Recursively enumerable	Turing machine
n/a	(no common name)	Recursive	Decider
Type-1	Context-sensitive	Context-sensitive	Linear-bounded
Type-2	Context-free	Context-free	Pushdown
Type-3	Regular	Regular	Finite

Each category of languages or grammars is a proper superset of the category directly beneath it.

Retrieved from "http://en.wikipedia.org/wiki/Finite_state_machine"

Categories: Computational models | Digital electronics | Formal methods